Real-Time Feature Tracking and Projective Invariance as a Basis for Hand-Eye Coordination

Gregory D. Hager

Research Report YALEU/DCS/RR-998
December 1993

Real-Time Feature Tracking and Projective Invariance as a Basis for Hand-Eye Coordination

Gregory D. Hager
Dept. of Computer Science
Yale University
P.O. Box 208285 Yale Station
New Haven, CT 06520-8285

E-mail: hager@cs.yale.edu
Phone: (203) 432-6432
Fax: (203) 432-0593

Abstract

This article describes a methodology for using *stereo visual feedback* to perform manipulation tasks. The two major innovations in the approach are: 1) the use of feature-based tracking methods that perform in real-time on standard workstations without specialized hardware; and 2) the use of closed-loop feedback control based on projective invariants to make servoing accuracy independent of calibration.

This article focuses on the construction and application of our feature tracking system. The system supports a variety of low-level detection methods (basic features), and features defined in terms of other features (composite features). We first describe how basic feature tracking has been implemented to be fast and robust on common workstations. We then develop a model of composite features as networks of state-based systems.

We present experimental results from two feature networks. One computes corresponding epipolar lines using eight corresponding features in two images. The second computes a visual trajectory for a visual servoing system.

Keywords: active and real-time vision, vision-guided robotics, experimental vision

To appear as a long paper in CVPR 1994

1 Introduction

This article describes a methodology for using *stereo visual feedback* to perform manipulation tasks. Although a great deal of research has been devoted to vision-based robotics, most vision-based systems are slow, unreliable, difficult to calibrate, expensive, and time consuming to build. One reason is that the data from vision must be processed efficiently so that it can be incorporated into a servo-loop that must execute in a timely fashion. The solution to this *data reduction* problem is usually problem specific and relies on specialized hardware. A second reason is that coordinating vision with a manipulator depends on knowing the transformation from visual to robot coordinates—the *hand-eye* transform. Most systems are extremely sensitive to errors in the hand-eye transform, making them inherently unreliable.

The main innovations in our approach are: the use of reconfigurable *window-based feature tracking* to simplify and increase the reliability of vision processing, the use of *feedback controllers based on projective invariants* to make system servoing accuracy independent of the hand-eye calibration, and the use of *geometry-based servoing strategies* to enhance reconfigurability. To be more concrete, consider tasks such as: constructing a tower of blocks, placing a grocery item into a box, picking up a coffee cup, inserting a floppy disk into a disk drive, or placing a wrench on a nut. All of these tasks can be broken down into a set of primitive operations which are defined in the visual domain. Each of these operations, which we henceforth refer to as *hand-eye skills*, has a geometric interpretation in the external world. Some example skills are:

Positioning: Move to a visual reference point.

Alignment: Orient along a visual reference line or align to a visual reference plane.

Motion: Move a point along a linear trajectory defined by a reference line.

Guarded Move: Move a set of reference features between or past other reference features.

For example, given stereo cameras, building a tower of blocks could be described in terms

of coordinated motions of features in both cameras as follows: move a corner of a block until it is on the line defined by an edge of the tower, align the two lines defining the face of the block with corresponding lines along the side of the tower, and move the block along these lines until touchdown. Given visually defined grasp points on the sides of a cup, grasping with a parallel jaw gripper could be executed as: align relative to grasp points, execute a guarded move toward the grasp points until the grasp points lie on the line defined by the two finger tips; maintain this position as fingertips close.

In this article, we give particular emphasis to the feature tracking component of the system. Historically, real-time vision has resorted to specialized, often custom-made, hardware in order to process the huge amount of information available in video images. However, the speed of standard workstations continues to increase, prices continue to decrease, and multi-processor architectures are becoming more widely available and accessible. These advances anticipate the day when many vision applications that do not require massive data transfers (*e.g.* full frame image processing) can be run on standard workstations outfitted with a simple framegrabber. Our tracking system is designed be a substrate from which such applications can be built. We have found that having a general-purpose "tracking tool" has enabled us to quickly assemble a wide variety of vision-based robotic applications in both the manipulation and mobile robot navigation domains.

The remainder of this document is organized as follows. The next section briefly compares our work with previous research in visual tracking and hand-eye coordination. Section 3 develops methods for accurate, calibration independent positioning and describes a methodology for expressing geometric tasks in terms of visual cues using ideas from projective geometry. Section 4 describes the tracking system in some detail. Section 5 describes two experiments with the system, and Section 6 indicates areas where we are improving the system.

2 Relationship to Previous Work

There has been a great deal of interest in non-feature-based, "attentional" or "reactive" vision systems for tracking. [13, 5, 6, 7, 37, 31]. However, most fine manipulation will require accurate template or feature-based tracking so that the visual servoing system can react to precise information about the configuration of the external world. Most reported feature-based tracking algorithms rely on global image processing followed by a feature-based correspondence solution. In some cases, the underlying representation is the pose of a known object in three dimensions, *eg.* [15, 28, 40]; in some cases the underlying representation is simply the 3-D location of features themselves, *eg.* [4, 26, 41]; and in some cases, the representation consists of geometric characteristics of the feature in the image [8, 9]. Most authors employ a linear smoother/predictor (*eg.* a Kalman filter) to update a feature or model information and to predict the location and/or appearance of features in images. Our research philosophy instead relies on using window-based image processing. The main advantages are speed, simplicity, and robustness. The latter comes from the fact that we can independently "tune" the image processing in each window to the particular feature or pattern that it is tracking. Similar real-time window-based processing appears in [10, 35]. Both systems rely on a specially constructed Transputer network to achieve real-time performance.

There is an abundance of literature related to visual servoing [25, 16, 1, 35, 3, 38, 34, 21, 24, 25, 33]. Systems take two forms: eye-in-hand or eye-on-arm systems where the camera is mounted on the manipulator, and hand-eye systems where the cameras are external to the controlled system. Nearly all hand-eye systems (cameras not on the manipulator) use vision to generate a Cartesian "reference trajectory" for the robot to track. This means that robot positioning is effectively open-loop with respect to vision. Closed-loop systems have been built by mounting camera(s) directly on the robot itself (*e.g* [30, 32, 38]). However, applying these systems to manipulation tasks still relies on an implicit or explicit hand-eye

calibration. Moreover, placing the camera on the robot restricts its viewing space, increases the likelihood that the robot will occlude the camera view, and decreases the payload of the robot.

Wijesoma *et.al.* [39] describe a closed-loop monocular hand-eye system for planar positioning using image feedback. In earlier work [17] we described a similar monocular visual servoing system and examined the use of adaptive control to compensate for calibration error. To our knowledge, the only stereo visual-servoing system similar in conception to ours is described in a recent paper by Hollinghurst and Cipolla [22]. There are three major dividing points between their work and ours. First, they use a locally valid affine approximation to the inverse perspective transformation requiring recalibration if the robot moves to a different point in the workspace. We use a perspective model that is globally valid. They compute the approximate Cartesian positions for both the end-effector and the goal position and compute control signals based on the difference between these positions. We eschew any sort of reconstruction and work purely in the visual domain. Finally, they describe a special purpose application of these concepts whereas we believe there are more general principles and a wider variety of hand-eye skills that can be elucidated.

Hutchinson [24] has developed a planning methodology for vision-based systems. However, his work has been restricted to monocular hand-on-arm systems using a single visual alignment primitive.

3 A Methodology for Visual Servoing

Recently, there has been increasing interest in applying ideas from projective geometry to vision [29, 12, 11]. Of particular interest are *projective invariants.* Our experience indicates that controllers based on invariant properties perform with an accuracy that depends only on the physical configuration of the system. In short: *the accuracy of servo control is independent of system calibration.*

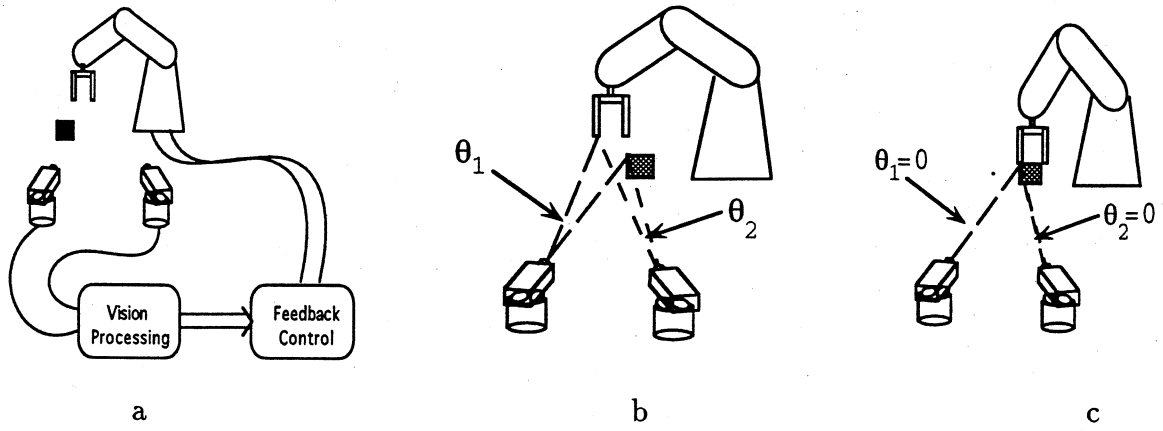

Figure 1. The figure on the left shows a visual servoing system consisting of two cameras on pan-tilt heads connected via a vision-based controller to a robot arm. The middle and right figure demonstrate positioning by reducing visual disparity to zero.

Figure 1(a) illustrates this idea for positioning. The left figure is a schematic depiction of our visual servoing architecture which includes two video cameras on pan-tilt heads, a robot arm, and computers that perform image-processing, low-level control operations, and other interface-related functions. Any attempt to accurately calibrate this system is limited by system modeling errors, mechanical backlash, control inaccuracy, and so forth. Consequently, a system that reconstructs absolute point coordinates using vision and positions the robot based on that information will be extremely imprecise. However, as illustrated in Figures 1(b) and 1(c) the robot manipulator can be positioned extremely accurately *relative to the observed target.* In (b), the cameras observe the visual disparity θ_1 and θ_2 between a point on the gripper and the corner of the box. We know the following property: *zero disparity between the manipulator and the goal in both images means that they are at the same point in space.* This property is true independent of where the cameras are located relative to the robot[1]. Thus, if we have a stable controller to achieve zero visual disparity, we can position

[1]More precisely this statement is true modulo configurations where the goal or robot and the cameras are collinear.

accurately, even if the system is badly calibrated.

Such a controller can be derived as follows. Consider two cameras located at positions c_1 and c_2 relative to an arbitrary fixed coordinate system $\mathcal{W}$. The cameras carry lenses with focal lengths f_1 and f_2, respectively. For each camera define a coordinate system by three orthogonal unit vectors, $\boldsymbol{x}_i$, $\boldsymbol{y}_i$ and $\boldsymbol{z}_i$ where $\boldsymbol{z}_i$ points along the optical axis of camera i, $\boldsymbol{y}_i$ points downward relative to the image of camera i, and $\boldsymbol{x}_i = \boldsymbol{y}_i \times \boldsymbol{z}_i$. The vector r designates the position (relative to $\mathcal{W}$) of a control point on a robot controlled by a velocity vector u. We assume the dynamics of the robot are negligible.

Using a pinhole camera model, the observation of r is given by:

$$\phi = \left(f_1 \frac{(r-c_1)\cdot \boldsymbol{x}_1}{(r-c_1)\cdot \boldsymbol{z}_1}, f_1 \frac{(r-c_1)\cdot \boldsymbol{y}_1}{(r-c_1)\cdot \boldsymbol{z}_1}, f_2 \frac{(r-c_2)\cdot \boldsymbol{x}_2}{(r-c_2)\cdot \boldsymbol{z}_2}, f_2 \frac{(r-c_2)\cdot \boldsymbol{y}_2}{(r-c_2)\cdot \boldsymbol{z}_2} \right)^T = g(r) \qquad (1)$$

Define $\dot{r} = u$ and let $\phi^* \in \Re^4$ denote the desired or "setpoint" value of ϕ. Also, introduce the error $e_\phi = \phi - \phi^*$ as well as the error integrating subsystem $\dot{z} = e_\phi$. We will assume ϕ^* has been chosen so that it is attainable—that is, there is a value r^* such that $\phi^* = g(r^*)$.

The control synthesis task is to choose u as a function of x, z, e_ϕ to stabilize the system

$$\dot{z} = e_\phi \qquad (2)$$

$$\dot{e}_\phi = J(r)u \qquad (3)$$

where $J(r)$ is the Jacobian of the composition $g(r)$. Assuming $J(r)$ has rank 3, stabilization can be carried out by setting

$$u = -(J(r)^T J(r))^{-1} J^T(r)(k_1 e_\phi + k_2 z) \qquad (4)$$

where k_1 and k_2 are positive gain constants chosen based on desired system behavior. We note that the problem can also be formulated so that the generalized inverse used above is not necessary [19].

Since the reference point r may be any point rigidly attached to the robot, its value is unknown and we must estimate it. The estimate, $\hat{r}$, is computed by rewriting the original system g in the form $A(\phi)r = b(\phi)$ and then defining

$$\dot{\hat{r}} = u - A^T(\phi)(A(\phi)\hat{r} - b(\phi)) \tag{5}$$

The estimate of r is used in (4). The stability analysis of the system appears in [19]. J and A have rank 3 as long as the cameras are spatially separated and the trajectory of r does not cross the line joining the optical centers of the camera. Thus, desired tracking will be achieved even if the camera calibration parameters are only approximately correct, provided the approximation errors are sufficiently small.

Given two reference vectors (a point and a line through the point) on the robot, it is also possible to control orientation by alignment with reference vectors in the world. We are currently developing a control algorithm based on this principle. The algorithm is more complex due to a larger number of unknown parameters and the nonlinearity of the problem. Initial simulation results seem to indicate that the same controller structure will work, however we have no stability analysis at this time.

3.1 Projective Geometry and Visual Servoing

Applying these control algorithms to perform a given task requires a *visual specification* of the task. That is, a geometric operation can only be accomplished by choosing visual features of the robot arm or payload, and visual features in the world to which some variant of the above control strategies can be applied. We now describe some of the geometric constructions that can be used to compute setpoints for positioning or alignment . Many of these ideas are based on recent progress in the area of "uncalibrated" vision using projective invariants [29, 11]. We recall the following facts:

- As already noted, except for certain singular configurations, two points are coincident if and only if their perspective projections in two or more images are coincident.

- Collinearity of points is preserved under projection. Modulo a set of singular configurations, two lines are collinear if and only if the projections in two camera images are are collinear.

- The projections of parallel lines meet at a single point in an image.

- Given (homogeneous) projections, p_l and p_r, of a point P in two images, there is a 3 by 3 matrix E such that $p_l^T E p_r = 0$. The matrix E can be computed from the projections of eight points in two images provided the points are arranged in a nonsingular configuration [27, 23, 29]. Hence, given a ninth point in one image, it is possible to determine the epipolar line along which the corresponding projection lies in the second image.

By applying geometric constructions based on the properties listed above, it is possible to use positioning and alignment to define many other "hand-eye skills." For example, it is often convenient to station the manipulator at some point in space *relative* to an object. Given the projection of a reference line in one image, we can choose an arbitrary point along that line as a stationing point. Using the E matrix, we can compute the corresponding epipolar line in the second image. The intersection of the projection of the reference line in the second image with the epipolar line is the location of the correct stationing point in the second image. This defines ϕ^* for *relative positioning along a line.* Given two reference points ϕ_1 and ϕ_2, we can define a time-varying reference point $\phi^*(t) = \phi_1 + t(\phi_2 - \phi_1)$. With minor modifications the controller can track these setpoints thereby achieving calibration independent *straight-line motion.* Given two parallel lines, we can compute the intersection point of their projections in two images.[2] Given a feature point in one image, we can compute the line through the intersection point and this feature point. Performing the same

[2]If they do not intersect, the camera is parallel or perpendicular to the lines. In the former case, and we take the projection to be at infinity, in the latter case we can do nothing.

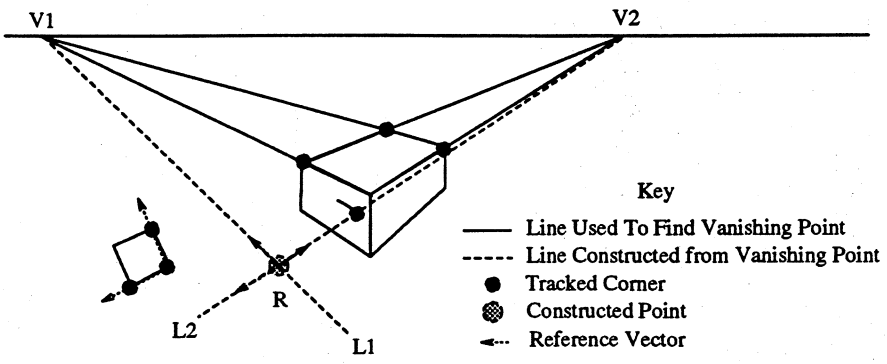

Figure 2. The geometric constructions for inserting a floppy disk into a drive unit.

construction for corresponding feature point in the second image yields a setpoint for *parallel alignment.*

Figure 2 describes the use of these geometric constructions to place a floppy disk in a disk drive. Geometrically, the strategy is to move the disk into the plane of the disk drive unit, align the edge perpendicular to the slot, and then move toward the slot until the disk slides in. In the visual domain, we can perform this task if we know that the sides of the disk drive are parallel to the floppy disk drive, the front of the unit is perpendicular to it, and the floppy itself forms a right angle. We first construct vanishing points $V1$ and $V2$ by tracking the indicated corners of the drive unit. The edge of the floppy slot and the vanishing point is used to construct the line $L2$. A reference point R along $L2$ is chosen, and the line $L1$ is constructed from the vanishing point $V1$. R, $L1$ and $L2$ define reference vectors (the setpoint) for alignment. Three corners of the floppy are used to construct the corresponding reference vectors on the disk itself. We note that the corresponding point for R in the second image would be computed as the intersection of $L2$ in the second image with the epipolar line corresponding to R. Finally, R can be moved along $L2$ to insert the floppy.

From this example, it is clear that hand-eye coordination depends heavily on an ability to locate and track specific features in an image.

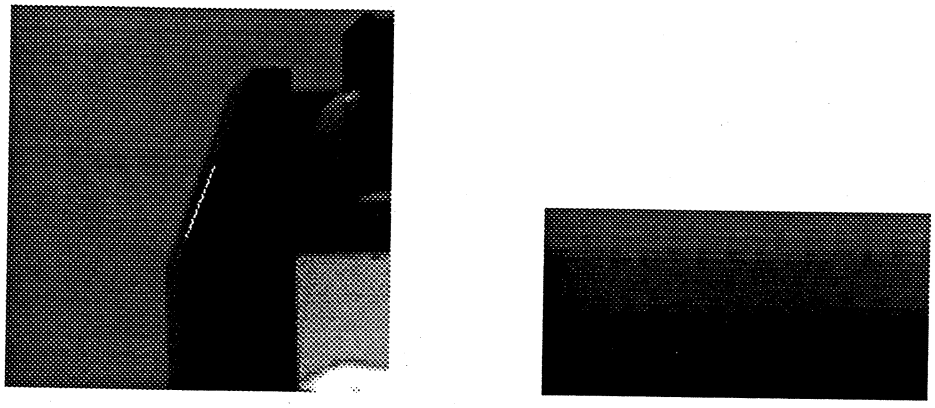

Figure 3. On the left, a sample image showing a reference line on the waist of the robot, and on the right the image associated with the window as it appears in window local coordinates.

4 Tracking System Design

Complex hand-eye tasks require a feature tracking system that can handle multiple features, can enforce various constraints among features, and can be simply reconfigured for different tasks and operating conditions. Our tracking system is based on two central ideas: window-based image processing, and state-based programming of networks of tracked features.

A window is an image defined by its height, width, position, and orientation in device (framebuffer) coordinates. All image processing operations within the window are defined *relative to the local window coordinate system.* Referring to Figure 3, suppose that we have an image with a window located about the white line in Figure 3(left). Then the image associated with that window (in window coordinates) is shown in Figure 3(right). Note that the line appears roughly horizontal in window coordinates.

Tracking a feature means that a window maintains a fixed, pre-defined relationship to the feature. Hence, any operation such as line detection or feature matching can be implemented assuming the requisite feature only deviates slightly from a standard orientation and position in local coordinates. This makes image processing simple to implement, fast to execute, and easy to specialize. At the same time, acquiring windows at any position

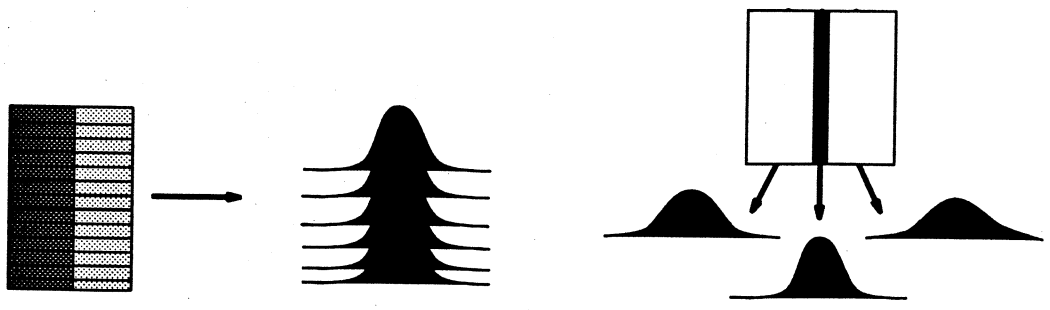

Figure 4. One dimensional convolution followed by superposition.

and orientation can be implemented quickly using ideas for fast rendering of lines and boxes borrowed from graphics [14]. Thus, the combination of movable oriented windows and image processing assuming a canonical configuration leads to fast feature tracking. The low-level features currently available in our system include solid or broken contrast edges detected using convolutions, and general grey-scale patterns tracked using SSD methods [2, 36]. These basic features can be easily composed into a wide variety of more complex configurations. The implementation of these methods is briefly described below.

4.1 Low-Level Operations

In most image processing systems, the majority of the time needed to perform edge detection is devoted to convolution. The key idea in fast contour localization is to use prior information to simplify detection. In window coordinates, detecting a contrast step edge in canonical position can be thought of as a series of one-dimensional problems as shown in Figure 4(left). Assuming the edge is vertical in the window, convolving each row of the window with a derivative-based kernel will produce an aligned series of responses. These responses can be superimposed by summing down the columns of the window. Finding the maximum value of this response function localizes the edge. If the edge is not correctly oriented, the response curve broadens. However, if the edge is symmetric, the maximum still correctly indicates the location of the edge in the window. Thus, for the price of a single pass with a one-dimension

convolution and a series of summations we can localize the position of a contrast edge.

In order to compute orientation, we sum the response to the scalar convolution along slanted lines as indicated in Figure 4 (right). The result is three response curves with different maximum values. The curve with the highest response is that closest in orientation to the underlying edge. By performing parabolic interpolation of the three curves, it is possible to predict the actual orientation of the underlying edge. In the ideal case, if the convolution template is symmetric and the response function after superposition is unimodal, the horizontal displacement of the edge should agree between all three filters. In practice, the estimate of edge location will be biased. For this reason, edge location is computed as the weighted average of the edge location of all three peaks. Assuming the convolution template can be expressed with integers, this entire operation can be performed with only integer addition and multiplies except for the interpolation step.

A particularly simple detector is a convolution template consisting of n -1's followed by m 0's followed by n 1's [18]. This kernel is attractive because no multiplications are needed to compute the convolution. Furthermore, the time to compute the convolution can be made independent of the size of the kernel. This is accomplished by noting that the difference in response between one pixel and the next can be computed by four additions and subtractions of pixels at the transition points of the convolution template. This detector on a Sun Sparc 2 with an Imaging Technologies 100 series framegrabber requires 1.5ms for a 20 pixel line searching $\pm$ 10 pixels using a mask 15 pixels wide.

There are several minor variations on this scheme. For example, the description above assumes that the convolution output has a constant sign. If the contour changes from a dark-to-light transition to a light-to-dark transition as we move along it, the superposition will yield a low response. Performing an absolute value operation after the convolution solves this problem.

Detectors based on these ideas have extremely good noise-rejection characteristics. Edges

outside the window do not affect their operation. Edges within the window, but oriented incorrectly do not cause much filter response and are generally rejected. In addition, the value and/or sign of the response can be used to enhance the "tuning" of the filter for a particular contour.

4.2 Patterns

We track arbitrary gray-scale patterns using SSD methods [2, 36]. To review this method, we denote the brightness of a pixel at location (x, y) at time t by $I(x, y, t)$. For the case of translation only, we formulate the SSD objective function in a window $\mathcal{W}$ as

$$\sum_{\mathcal{W}}(I(x + u\delta t, y + v\delta t, t + \delta t) - I(x, y, t))^2$$

Expanding this expression in a Taylor series yields

$$\sum_{\mathcal{W}}(I_x u + I_y v + I_t)^2.$$

Minimizing by taking derivatives and rearranging yields a linear system:

$$H(I_x, I_y)\begin{bmatrix} u \\ v \end{bmatrix} = b(I_x, I_y, I_t).$$

The matrix H will be full rank only if the local gradient directions over the window span the plane. In order to deal with so-called *aperture problems,* we compute the vector $(u, v)^T$ using the psuedo-inverse, denoted $^{\#}$:

$$\begin{bmatrix} u \\ v \end{bmatrix} = H^{\#} b$$

Variations on this method include parameters for scale and orientation. This tracking method requires 70 ms to localize a 20 by 20 window on a Sun Sparc 2. This operator is usually applied at multiple resolutions to increase the range of motions that it can track.

4.3 Networks of Features

Every feature or image property in our system can be characterized in terms of a state vector. For *basic features*—those that operate directly on images—the state of the feature tracker is usually the position and orientation of the feature (= that of its window) relative to the framebuffer coordinate system. We define *composite features* to be features that compute their state from other basic and composite features.

To illustrate the relationship between basic and composite features, consider computing the location of the intersection of two contours. The state of a single contour tracker in an image is the vector $L = (x, y, \theta)^T$ describing the location of a window centered on the contour and oriented along it. The low-level feature detection methods described above compute an offset normal to the edge, δt, and an orientation offset $\delta\theta$. Given these values, the state of the contour tracker is updated according to the following equation:

$$L^+ = L^- + \begin{bmatrix} -\delta t \sin(\theta + \delta\theta) \\ \delta t \cos(\theta + \delta\theta) \\ \theta + \delta\theta \end{bmatrix} \tag{6}$$

Note that we have an aperture problem: the state vector, L, is not fully determined by information returned from feature detection. There is nothing to keep the window from creeping "along" the contour it is tracking.

This problem can be solved by defining a composite feature that is the intersection of two non-colinear contours. This feature has a state vector $C = (x, y, \theta, \alpha)^T$ describing the position of the intersection point, the orientation of one contour, and the orientation difference between the two contours. From image contours with state $L_1 = (x_1, y_1, \theta_1)^T$ and $L_2 = (x_2, y_2, \theta_2)^T$, the distance from the center of each tracking window to the point of intersection the two contours can be computed as

$$\begin{aligned} \lambda_1 &= ((x_2 - x_1)\sin(\theta_2) - (y_2 - y_1)\cos(\theta_2))/\sin(\theta_2 - \theta_1) \\ \lambda_2 &= ((x_2 - x_1)\sin(\theta_1) - (y_2 - y_1)\cos(\theta_1))/\sin(\theta_2 - \theta_1) \end{aligned}$$

The state of a corner $C = (x_c, y_c, \theta_c, \alpha_c)$ is calculated as:

$$\begin{aligned} x_c &= x_1 + \lambda_1 \cos(\theta_1) \\ y_c &= y_1 + \lambda_1 \sin(\theta_1) \\ \theta_c &= \theta_1 \\ \alpha_c &= \theta_2 - \theta_1 \end{aligned} \tag{7}$$

Given a fixed intersection point, we can now *choose* "setpoints" λ_1^* and λ_2^* describing where to position the contour windows relative to the intersection point. With this information, the states of the individual contours can be adjusted as follows:

$$\begin{aligned} x_i &= x_c - \lambda_i^* \cos(\theta_i) \\ y_i &= y_c - \lambda_i^* \sin(\theta_i) \end{aligned} \tag{8}$$

for $i = 1, 2$. Choosing $\lambda_1^* = \lambda_2^* = 0$ defines a cross pattern. If the window extends h pixels along the contour, choosing $\lambda_1^* = \lambda_2^* = h/2$ defines a corner. Choosing $\lambda_1^* = 0$ and $\lambda_2^* = h/2$ defines a tee, and so forth.

A complete tracking cycle for this system would consist of first computing (8) to make the initial state of the contours consistent, then performing low-level feature detection, and finally computing (6) followed by (7).

More generally, we define a *feature network* to be a set of nodes connected by two types of directed arcs referred to as *up-links* and *down-links*. Nodes represent basic and composite features. Up-links represent the information dependency between a composite feature and the features used to compute its state. Thus, if a node is a source node with respect to up-links, it must be a basic feature. If a node has incoming up-links it must be a composite feature. If a node n has incoming up-links from nodes $m_1, m_2, \ldots, m_k$ the latter are called *subsidiary nodes* of n. Down-links represent the imposition of constraints or other high-level

information on features. A node that is a source for down-links is a *top-level node.* If a node n has incoming down-links from nodes $m_1, m_2, \ldots m_k$, the latter are called *supersidiary nodes* of n. All directed paths along up-links or down-links in a feature graph must be acyclic. We also require that every top-level feature that is path-connected to some basic feature with respect to up-links must be path-connected to the same basic feature via down-links. For example, a corner is a graph with three nodes. The corner feature is a top-level feature. The two contours which compose it are subsidiary features. There are both up-links and down-links between the corner node and the feature nodes.

Given this terminology, we can now define a complete tracking cycle to consist of: 1) traversing the down-links from each top-level node applying state constraints until basic features are reached; 2) applying low-level detection in every basic feature; and 3) traversing the up-links of the computing the state of composite features. We note that state prediction can be added to this cycle by including it in the downward propagation.

5 Example Applications

In this section, we briefly describe a programming environment for developing feature networks, and then describe the results of applying this environment to two problems: correspondence calculation and hand-eye servoing.

5.1 Programming Environment

We have constructed a programming environment in C++ to facilitate the construction of feature-tracking networks. The system implements a subset of feature networks in which each node may have only one supersidiary node. This means that feature graphs will be trees in which each up-link from node n to node m has a corresponding down-link from m to n.

Basic features are derived from a base class called `Feature`, and composite features are derived from a base class called `CFeature`. Any variable of type `Feature` provides a function for updating state from image information, and a display function. A variable of type `CFeature` is a `Feature` that maintains an internal list of subsidiary `CFeatures` as well as its own internal state. Information is propagated up and down the feature network using two functions: `compute_state` which computes a composite feature's state from the state of its subsidiary nodes, and `state_propagate` that adjusts the state of a subsidiary nodes based on the state of their supersidiary node. The default update cycle for a `CFeature` is to call its own `state_propagate` function, to call the update function of the children, and then to call `compute_state`. This is combined into a single function `refresh()` callable only from a top-level feature. Calling it sweeps information down the network to the set of basic features, updates of the state of all basic features, and sweeps updated state information back up the network.

We have found that this programming environment greatly facilitates the development of tracking applications, and leads to clear compact program semantics. For example, a contrast edge tracker and a corner tracker can be defined as:

```
Video v(1);  Edge e; // Open a video device 1 and declare an edge detector
Line l(&e,&v);       // Declare a contour tracker in video device 1
Corner c(&e, &v);    // Declare a corner tracker in video device 1
```

A corner is actually a `CFeature` that internally allocates two edge trackers, adds them to a queue of objects, and manages them using the constraint functions described above. Both features can be added to a container `Cfeature` variable:

```
CFeature p;        // Declare a composite feature with no constraint functions
p += l;  p += c;   // add features to internal queue.
```

Once all features are properly initialized, the main loop of any tracking application is of the form:

```
  while (...) {
   p.refresh();
   ... other user code ...
}
```

Naturally, any other user code impacts the speed of tracking and so must be limited to operations that can be performed in a small fraction of a second. Adding a new composite feature to the system involves deriving a class from `CFeature`, writing the constraint functions, and defining the subsidiary features that the class depends on [20].

5.2 Tracking Corresponding Epipolar Lines

We have already described how construction of the epipolar line corresponding to a feature in an image is an essential tool for computing visual setpoints. Given movable cameras, tracking epipolar lines also makes it possible to establish additional point correpondences. The procedure is as follows: 1) hold the first camera still; 2) locate a new feature in the second camera image; 3) compute the corresponding epipolar line in the first image; 4) move the second camera; 5) compute another epipolar line in the first image. The corresponding feature point in the first camera is at the intersecton of two epipolar lines. We note this is effectively the same technique as is used in trinocular stereo to disambiguate correspondences.

Briefly, the E matrix can be computed by locating $n \geq 8$ corresponding points in two images, compiling an n by nine matrix A, and selecting the eigenvector corresponding to the smallest eigenvalue of $A^T A$ [23]. The nine values of the eigenvector are the nine entries in E. Figure 5 describes the feature network used to implement corresponding epipolar line tracking. We rely on corners as defined in Section 4.3. We define the class `CorFeature` which manages two features in separate video images. Its state is the x and y locations of both features in their respective images. It imposes no constraints on the state of subsidiary features. Next, we define the class `EMatrix` to be a `CFeature` that manages eight or more corresponding features. It contains a `compute_state` function that computes the E matrix

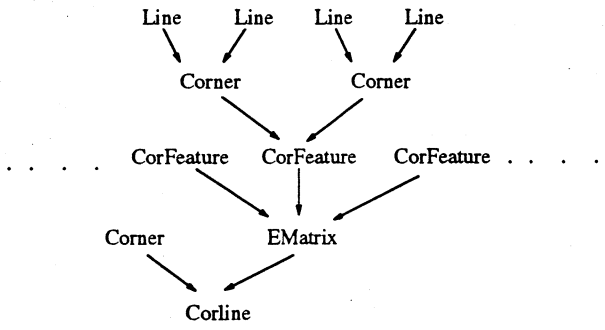

Figure 5. The feature network used to compute corresponding epipolar lines

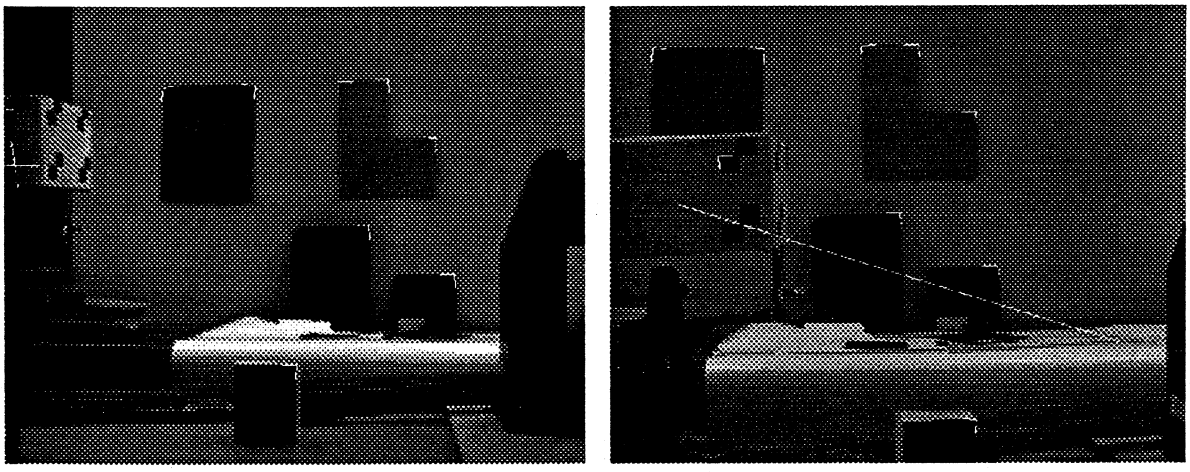

Figure 6. Left, the first image with a chosen feature (the cross at the far left) and right, the second image showing the corresponding epipolar line.

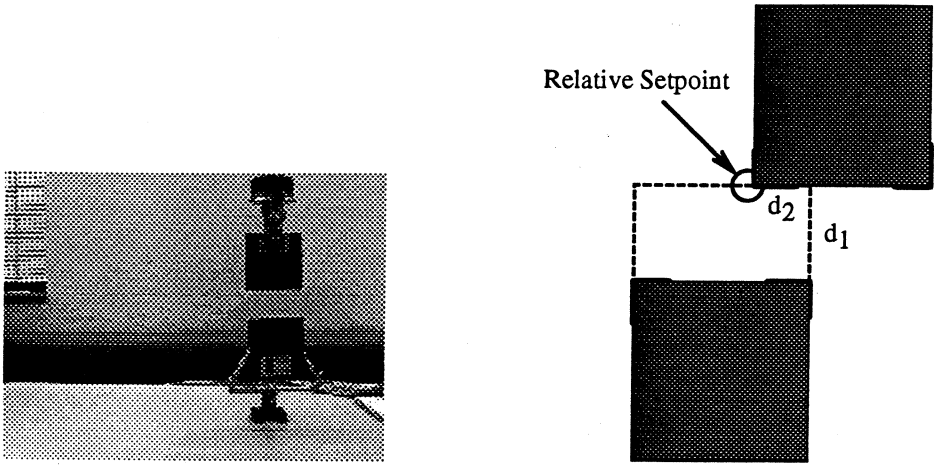

Figure 7. An image of the servoing system as seen from the camera, and a diagram of how the trajectory was defined.

as described above. Finally, we define a class `Cline` that manages a corner and an instance of an `Ematrix`. It has a `compute_state` function that computes the equation of the epipolar line in the second image. The display function is defined to show the line in the second video image.

Figure 6 shows two images from an experimental run of the system. Using nine corners composed of features 12 pixels long, we were able to reliably compute and track the corresponding epipolar line for a tenth feature point (the hash pattern on the lower right of the box) at a rate of approximately 10 Hz. The error in correspondence location was typically less than five pixels. It was interesting to note that the epipolar line equations are actually quite noisy. However, the computed line "swivels" about the corresponding feature point. Thus, while the line itself may swing as much as 20 or 30 pixels in some regions, it is extremely stable at the corresponding feature point.

5.3 Visual Servoing Experiments

We have implemented a system for position control using the controller described in Section 3. The system consists of a Sparc II computer that performs image processing and control calculations, and a PC that controls our Zebra Zero robot. Two cameras mounted on pan-tilt heads located approximately 1 meter from the robot, 30 cm. apart observe the workspace of the robot. One test consists of tracking the corners of two 3 1/4 inch floppy disks and executing a rectangular path that starts by touching two corners of the disks, moves to touch alternate corners, and then returns to the original configuration.

This trajectory is described as follows. We define a class `Rect` that manages two corners connected by a straight contour. The state of the `Rect` composite feature depends on two constants, d_1 and d_2 as well as the positions of the two corners which we will denote c_1 and c_2. Let p denote a vector of length $\|c_2 - c_1\|$ perpendicular to the line joining c_1 and c_2—that is, $p \cdot (c_2 - c_1) = 0$. Then position of the setpoint, s is given by

$$s = c_1 + d_2(c_2 - c_1) + d_1 p.$$

(see Figure 7). We then define a class `SRect` that maintains a stereo pair of `Rects` having the same values of d_1 and d_2. The state vector of `SRect` is the concatenation of the state variables of its children. Changing the setpoint values of an `SRect` effectively describes a *stereo* trajectory for the robot to track[3]. Two instances of this class are used to track the bottom of the floppy disk held by the robot of the target floppy in the stand. The state variables of this system are the robot visual position (ϕ above) and the robot setpoint value (ϕ^* above). Executing the test trajectory requires changing the setpoint values of both the robot disk tracker and the target disk tracker though eight different motion segments.

When tracking contours 16 pixels long in a region of ± 10 pixels the tracking system alone runs at 25 Hz. With robot control in the loop, the system runs at 20 Hz. With the physical configuration as described above, this implies that the robot can be run with endpoint speeds

[3]This approach assumes the camera imaging planes are roughly parallel to the face of the disk. In general, we would have to use an epipolar construction to compute corresponding setpoints in two images.

of up to approximately 12 cm/sec. Assuming contour localization is accurate to 0.5 pixels, the expected error in localization is about ± 1.25mm. The width of the disk is exactly 2.5mm, so we expect the robot to be able to actually touch the corners of the disks together. In every trial, the system achieved position accuracies well within expected tolerances: less than 0.2mm in the direction parallel to the camera imaging planes and approximately ± 1.0mm in depth. The system is also extremely stable and insensitive to calibration. During operation, we can move the cameras several centimeters and rotate them several degrees with little apparent effect on the speed and accuracy with which setpoints are attained. More detailed comments on this system can be found in [19].

6 Future Work

In this article, we have described a novel approach to hand-eye coordination using closed-loop visual servoing. We have shown how a geometric task can be decomposed a set of visual reference points and visual trajectories. Implementing vision-based tasks requires a system that can quickly and accurately track a variety of features. We have constructed such a system and demonstrated its use for epipolar line calculation and for hand-eye servoing.

There are several areas where we are working to improve and extend the theory and practice of building tracking-based servoing systems. The major limitations of the current tracking system is its lack of error-handling capability. We are investigating methods for improving the low-level rejection of spurious features, and we are looking into methods from robust statistics to improve the system's tolerance to momentary mistracking. We would also like to develop a means of stability analysis for feature tracking networks.

We are currently designing and implementing a controller for alignment. A preliminary design appears to work in simulation and will be moved to the hand-eye system in the near future. In the longer term, we plan to construct a larger variety of "hand-eye skills" using geometric constructions and feedback control.

Acknowledgements The work in this paper could not have been done without the help of Sidd Puri on implementing an initial version of the tracking system, and W-C. Chang and A.S. Morse for help in designing the visual feedback controller. This research was supported by DARPA grants N00014-91-J-1577 and N00014-93-1-1235, and by National Science Foundation grants IRI-9109116 and DDM-9112458.

References

[1] P. Allen, B. Yoshimi, and A. Timcenko. Real-time visual servoing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 851–856. 1991.

[2] P. Anandan. A computational framework and an algorithm for the measurement of structure from motion. *Int. J. Computer Vision*, 2:283–310, 1989.

[3] R. L. Anderson. Dynamic sensing in a ping-pong playing robot. *IEEE Journal of Robotics and Automation*, 5(6):723–739, 1989.

[4] N. Ayache and O. Faugeras. Maintaining representations of the environment of a mobile robot. *IEEE Trans. on Robotics and Automation*, 5(6):804–819, December 1989.

[5] P. Burt, J. Bergen, R. Hingorani, R. Kolczinski, W. Lee, A. Leung, J. Lubin, and H. Shvaytser. Object tracking with a moving camera: An application of dynamic motion analysis. In *Proceedings of the Workshop on Visual Motion.* 1989.

[6] J. Connell. *A Colony Architecture for an Artificial Creature.* PhD thesis, MIT, 1989. Technical Report 1151.

[7] D. Coombs and C. Brown. Real-time smooth pursuit tracking for a moving binocular head. In *Proc. IEEE Conf. on Comp. Vision and Patt. Recog.* 1992.

[8] J. L. Crowley and P. Stelmaszyk. Measurement and integration of 3-D structures by tracking edge lines. In *European Conference on Computer Vision*, pages 269–280. 1990.

[9] R. Deriche and O. Faugeras. Tracking line segments. *Image and Vision Computing*, pages 261–270, 1991.

[10] E. D. Dickmanns and V. Graefe. Dynamic monocular machine vision. *Machine Vision and Applications*, 1:223–240, 1988.

[11] O. Faugeras. What can be seen in three dimensions with an uncalibration stereo rig? In *Computer Vision-ECCV '92*, pages 563–578. Springer Verlag, 1993.

[12] O. Faugeras, Q.-T. Luong, and S.J.Maybank. Camera self-calibration: Theory and experiments. In *Computer Vision-ECCV '92*, pages 321–334. Springer Verlag, 1993.

[13] A. M. Flynn and R. A. Brooks. MIT mobile robots — what's next? *IEEE Journal of Robotics and Automation(?)*, pages 611–617, 1988.

[14] J. Foley, A. v. Dam, S. Feiner, and J. Hughes. *Computer Graphics.* Addison Wesley, 1993.

[15] D. Gennery. Tracking known 3D objects. In *Proc. Am. Assoc. Art. Intell.*, pages 13–17. 1982.

[16] D. Gershon and I. Porat. Vision servo control of a robotic sewing system. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1830–1835. 1988.

[17] G. Hager. Some problems in adaptive visual servoing. DCS RR-948, Yale University, New Haven, CT, January 1993.

[18] G. D. Hager. *Task-Directed Sensor Fusion and Planning.* Kluwer, Boston, MA, 1990.

[19] G. D. Hager, W.-C. Chang, and A. S. Morse. Robot feedback control based on stereo vision: Towards calibration-free hand-eye coordination. DCS RR-992, Yale University, New Haven, CT, October 1993. Submitted to the 1994 International Conference on Robotics and Automation.

[20] G. D. Hager, S. Puri, and K. Toyama. A framework for real-time vision-based tracking using off-the-shelf hardware. DCS RR-988, Yale University, New Haven, CT, September 1993.

[21] K. Hashimoto, T. Kimoto, T. Ebine, and H. Kimura. Manipulator control with image-based visual servo. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2267–2272. 1991.

[22] N. Hollinghurst and R. Cipolla. Uncalibrated stereo hand eye coordination. Technical Report TR-126, Cambridge University, Dept. of Engineering, September 1993.

[23] T. S. Huang and C. H. Lee. Motion and structure from orthographic projection. *IEEE Trans. Pattern Anal. Mach. Intelligence*, 11(5):536–540, May 1989.

[24] S. Hutchinson. Exploiting visual constraints in robot motion planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1722–1727. 1991.

[25] S. T. Jiang Yu Zheng, Qian Chen. Active camera guided manipulation. In *Proc IEEE Int. Conf on Robotics and Automation*, pages 632–638. Sacramento, 1991.

[26] J. J. Leonard, H. F. Durrant-Whyte, and I. J. Cox. Dynamic map building for an autonomous mobile robot. *Int. J. of Robotics. Res.*, 11(4):286–298, 1992.

[27] H. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.

[28] D. G. Lowe. Robust model-based motion tracking through the integration of search and estimation. *Int. J. Computer Vision*, 8(2):113–122, 1992.

[29] J. Mundy and A. Zisserman. *Geometric Invariance in Computer Vision.* MIT Press, Cambridge, Mass., 1992.

[30] B. Nelson and P. K. Khosla. Increasing the tracking region of an eye-in-hand system by singularity and joint limit avoidance. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 418–423. 1993.

[31] R. Nelson and J. Aloimonos. Obstacle avoidance: Towards qualitative vision. In *Proc. Int'l Conf. on Computer Vision.* 1988.

[32] N. Papanikolopoulos, P. Khosla, and T. Kanade. Visual tracking of a moving target by a camera mounted on a robot: A combination of control and vision. *IEEE Transactions on Robotics and Automation*, 9(1), 1993.

[33] N. Papanikolopoulos, P. Khosla, and T. Kanade. Vision and control techniques for robotic visual tracking. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 857–864. 1991.

[34] P. Rives, F. Chaumette, and B. Espiau. Positioning of a robot with respect to an object, tracking it and estimating its velocity by visual servoing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2248–2254. 1991.

[35] A. Rizzi and D. E. Koditschek. Further progress in robot juggling: The spatial two-juggle. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 919–924. 1993.

[36] C. Tomasi and T. Kanade. Shape and motion from image streams: a factorization method, full report on the orthographic case. CMU-CS 92-104, CMU, 1992.

[37] R. Wallace et. al. The lunchbox vision system. Vision Colloquium at Yale University., October 1991.

[38] L. Weiss, A. Sanderson, and C. Neuman. Dynamic sensor-based control of robots with visual feedback. *IEEE J. of Robotics and Automation*, RA-3(5):404–417, Oct. 1987.

[39] S. Wijesoma, D. Wolfe, and R. Richards. Eye-to-hand coordination for vision-guided robot control applications. *Int. J. Robot. Res.*, 12(1):65–78, 1993.

[40] J. Wu, E. Ring, and e. al. Recovery of the 3D location and motion of a rigid object through camera image (an extended kalman filter approach). *Int. J. Computer Vision*, (3):373–394, 1988.

[41] Z. Zhang and O. Faugeras. A 3d world model builder with a mobile robot. *Int. J. of Robotics Res.*, 11(4):269–285, 1992.